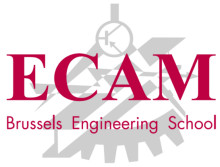
Ecole Centrale des Arts et Métiers



Rapport de stage d'immersion en entreprise de 2éme Master

Développement du firmware d'une smart-card réalisant des opérations cryptographiques

THALES

Vander Meiren Antoine
ECAM 2017 - 2018

Maitre de stage : ir. TESSIAN Samuel
Superviseur de stage : Dr ir. COMBEFIS Sébastien

Table des matières

Liste des figures

Liste des acronymes

AES Advanced Encryption Standard
ANS Autorité Nationale de Sécurité
APDU Application Protocol Data Unit
BMS Battlefield Management System
CBC Cipher Block Chaining
DES Data Encryption Standard
EEPROM Electrically-Erasable Programmable Read-Only Memory
IV Initialization Vector
JCDK Java Card Development Kit
JCRE Java Card Runtime Environnement
PKCS Public-Key Cryptography Standards
PKI Public Key Infrastructure
RAM Random Access Memory

Partie 1

Introduction

Du 15 septembre au 27 octobre 2017, j'ai effectué un stage d'immersion en entreprise au sein de Thales Belgium (situé à : Rue des Frères Taymans 28, 1480 Tubize). Au cours de ce stage dans le département InfoSec, j'ai pu m'intéresser particulièrement au monde de la cryptographie.
Plus largement, ce stage a été l'opportunité pour moi d'appréhender le déroulement de la vie en entreprise ainsi que de dégrossir mon travail de fin d'études.

Ce rapport présente l'entreprise ainsi que les différentes technologies utilisées durant le stage. Vous aurez également un aperçu des tâches effectuées, du projet qui m'a été confié et des objectifs accomplis pour finir avec une conclusion.

Remerciements

Je tiens à remercier toute l'équipe du département InfoSec de Thales Belgium, en particulier mon maître de stage Tessian Samuel, pour leur aide et le partage de leur expertise au quotidien. Je remercie également ma sœur, Vander Meiren Anne-Sophie, pour la relecture le mon rapport.

Partie 2

Présentation de l'entreprise

2.1 Vue globale

Le groupe Thales est présent dans 56 pays et emploie près de 64 000 personnes pour un chiffre d'affaires d'approximativement 15 milliards d'euros en 2016. [27] Le groupe est organisé en 5 divisions opérationnelles qui correspondent à 5 secteurs d'activité : la défense, la sécurité, l'espace, l'aéronautique et le transport terrestre. La figure 1 détaille l'importance de chaque division du groupe en 2014. Le rôle de l'entreprise est d'aider ses clients à remplir leur mission, grâce à une maîtrise de la chaîne de décisions critiques.

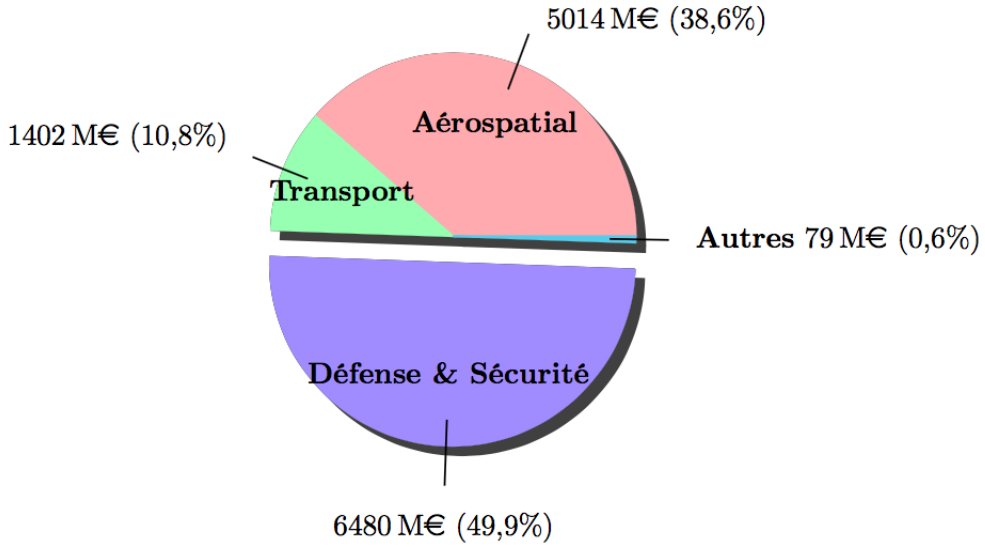


Figure 1: Chiffre d'affaires des pôles d'activité du groupe Thales en 2014 [15]

Le rôle du groupe inclut la fourniture des outils technologiques et des systèmes nécessaires au recueil de données, leur transmission sécurisée ainsi que leur traitement dans le but de faire les bons choix, afin de mieux gérer et réagir aux situations auxquelles les clients font face. En résumé, Thales aide ses clients à choisir la meilleure option et agir en conséquence.

L'histoire de Thales a commencé en 1893 avec la création de la compagnie Française Thomson-Houston, destinée à exploiter en France les brevets américains de la société Thomson-Houston Electic Compagny dans le domaine de la production et du transport de l'électricité. Après un rapprochement avec la compagnie générale de la télégraphie sans fil (CSF), la société diversifie ses activités et se rapproche notamment du monde de la communication téléphonique. [28]
En 2000, Thomson-CSF devient Thales et est alors une société centrée sur l'électronique de défense et les technologies de l'information. Suite à son rapprochement avec Alcatel-Lucent en 2007, la société étend son domaine d'activité au transport, à la sécurité et à l'espace, aboutissant à la création de Thales Alenia Space.

Le groupe Thales possède cinq implantations en Belgique. Thales Alenia Space ETCA, à Charleroi, est le leader mondial en conditionnement et distribution d'énergie à bord des satellites et le plus important fournisseur d'électricité pour Ariane 5. Implantée à Herstal, la filiale Forges de Zeebruges est une société orientée vers l'export et développant un système de roquette guidée d'une toute nouvelle génération. Cette entreprise est d'ailleurs la seule au monde à pouvoir maîtriser l'ensemble d'un système de roquette air-sol. Deux autres installations plus petites sont situées à Bruxelles. Finalement, Thales Belgium, implanté à Tubize en périphérie bruxelloise que nous détaillerons dans la section suivante. [17]

2.2 Thales Belgium

Thales Belgium s.a. (anciennement Thales Communications Belgium) emploie environ 130 personnes dont une cinquantaine d'ingénieurs. Son chiffre d'affaires annuel moyen est de 40 millions d'euros en 2015. Thales Belgium est le centre d'expertise belge du groupe Thales, internationalement reconnu pour le développement et la fourniture de système de défense et de sécurité. La société est active dans la conception, la fourniture et la maintenance des systèmes de communication critiques tel que : [14]

- Système de communication et d'information : intégration de moyens vétronique, BMS (Battlefield Management System) et communication, soldat monté, communication satellite.
- Communications terrestres tactiques et navales : boîtes d'antennes et amplificateurs de puissance pour radio HF, systèmes radio et coupleur d'antennes pour communication HF et VHF.
- Communication en aéronautique : système d'intercommunication pour hélicoptères effectuant des missions spéciales de surveillance ou de patrouilles maritimes, technologie VoIP, modems pour VHF Data Radio;
- Système de cryptographie en ligne et hors ligne approuvé par l'ANS (Autorité Nationale de Sécurité). [1]

Récemment, les projets Thales Belgium ont abouti à des équipements tels que la vétronique de l'armée belge ainsi que le système de géolocalisation des bus de la STIB, utilisé pour alerter en direct les navetteurs des retards. L'autonomie laissée à chacune de ces divisions par le groupe permet à Thales Belgium de fonctionner avec la flexibilité d'une PME, tout en bénéficiant de la solidité conférée par un grand groupe international. La société se structure en pôles de compétences (informatique embarquée, sécurité, conception mécanique, etc.) qui constituent pour chaque project une équipe pluridisciplinaire capable de prendre en charge l'ensemble du projet, du concept au prototypage et à l'industrialisation. [14]
En effet, disposant d'une ligne de production, les tests et la validation des produits conçus en engineering peuvent se faire dans la foulée de leur conception, et cela reste un grand avantage de l'implantation de Tubize.

La figure 2 reprend un organigramme de l'ensemble des divisions de Thales Belgium afin d'arriver jusqu'à la ligne de produit InfoSec pour laquelle travaille mon maitre de stage, Tessian Samuel. L'équipe InfoSec, à laquelle j'ai été intégré, est chargée du développement d'une ligne de produits dans le domaine du chiffrement, produits approuvés par l'ANS. Citons par exemple la gamme de chiffreur A891. [16]
Développant une ligne de produit complète pour le client, il est dès lors important de rester dans le contexte d'utilisation de ces produits, qui travaillent en synergie les uns avec les autres. Le travail de l'équipe se situe au carrefour entre les attentes des clients, la conception de produit répondant à ces attentes, les contraintes techniques de développement et l'environnement industriel de production. C'est ce qui rend le travail au sein d'une équipe avec de telles préoccupations très complet et enrichissant.

2.3 Enjeux

La sécurité informatique est l'enjeu principal de l'équipe InfoSec et par conséquent du projet qui m'a été attribué durant ce stage. Les produits de l'entreprise sont utilisés dans un contexte où la sécurité de l'information échangée est primordiale, il sera donc essentiel de veiller à mettre en place un système aussi fiable et sécurisé que possible.
Face à de telles exigences, il est nécessaire de justifier ses choix techniques et les comparer aux attentes de l'entreprise en matière de sécurité.

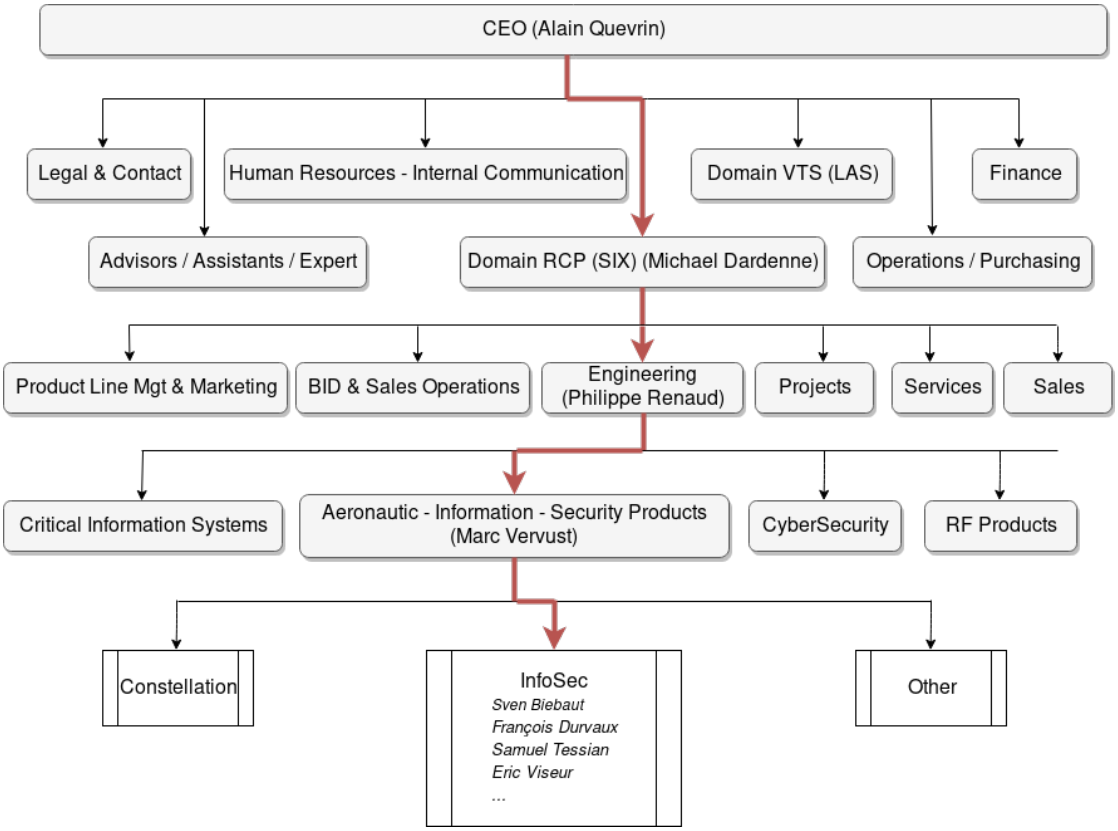


Figure 2: Organigramme Thales Belgium [18]

Nous travaillons donc dans le département Domain RCP SIX (Radio Communication Products, Secure Information and Communication Systems), et plus précisément dans l'engineering Aeronautic, Information and Security Products.

Partie 3

Cryptographie

Cette partie décrit quelques notions théoriques nécessaires à la compréhension des différentes tâches effectuées durant ce stage. Cette dernière décrit notamment le fonctionnement de base de la cryptographie, les besoins de Thales Belgium ainsi que la solution déjà proposée par l'entreprise.

3.1 Fonctionnement

Il existe deux grands types de cryptographie appelés "à clé symétrique" et "à clé asymétrique". Dans cette section, nous expliquerons brièvement le principe de fonctionnement de chacun d'eux.

3.1.1 Cryptographie à clé symétrique

La cryptographie symétrique utilise une clé secrète pour chiffrer et déchiffrer un message. En simplifiant le principe de fonctionnement mathématique, nous pouvons l'expliquer comme suit :

Une clé K permet d'obtenir une fonction de chiffrement e_k et de déchiffrement d_k. Il suffit alors d'appliquer la fonction e_k à un message M pour obtenir ce que l'on appelle un ciphertext (ou cryptogramme en français) ou encore la fonction d_k à un ciphertext pour en récupérer le message. [21]

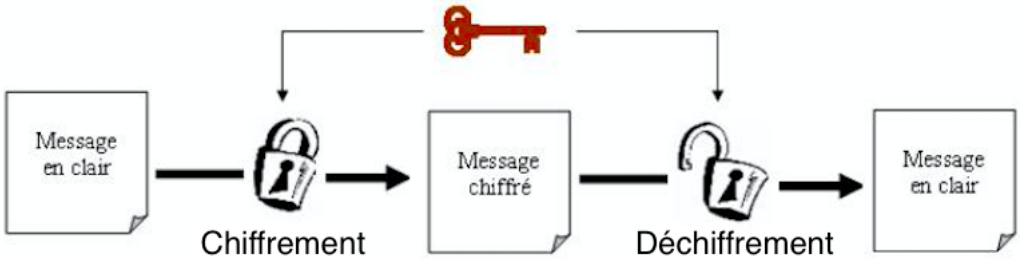


Figure 3: Fonctionnement de la cryptographie à clé symétrique [4]

Sur la figure 3, nous pouvons voir à gauche le message dit "en clair" donc compréhensible par un tiers. Ce message passe dans la fonction e_k nommée "Chiffrement" obtenue grâce une clé K afin d'en sortir le ciphertext nommée "message chiffré". Finalement nous passons le ciphertext dans la fonction d_k nommée "Déchiffrement" obtenue grâce à cette même clé K pour retrouver le message "en clair" initial.

Dans ce type de cryptographie, si un message doit être envoyé de manière sécurisée à une autre personne, les deux interlocuteurs doivent absolument au préalable s'être partagé cette clé secrète.
Dans le cadre du stage, nous pouvons nous affranchir de cette contrainte car les clés sont partagées à l'avance par une application existante dans l'entreprise.

Il existe encore une fois plusieurs algorithmes de chiffrement symétrique, notamment le chiffrement DES (Data Encryption Standard) et AES (Advanced Encryption Standard).

Chiffrement DES :

Cet algorithme fonctionne avec une clé de 56 bits (64 bits avec un bit de parité par octet) et un message découpé en bloc de 64 bits. Une série d'opérations/ permutations sera effectuée sur chaque bloc de message afin d'en obtenir un bloc de ciphertext. L'opération inverse sera effectuée pour récupérer le message initial. Aujourd'hui cet algorithme est fortement menacé par la puissance de calcul des ordinateurs qui peuvent facilement balayer la plupart des clés disponible pour retrouver le message ayant été chiffré.[5]

Chiffrement AES :

Prévu pour remplacer le chiffrement DES, l'AES est mathématiquement plus efficace que son prédécesseur mais sa force principale vient de la longueur variable de ses clés. En effet, AES permet l'utilisation de clé de 128, 192 ou encore 256 bits ce qui le rend exponentiellement plus résistant. [20]

Nous pouvons utiliser l'algorithme AES dans plusieurs modes de fonctionnement, par exemple le mode CBC (Cipher Block Chaining). Comme l'algorithme DES, AES fonctionne avec des blocs de données. Une donnée sera donc découpée en plusieurs morceaux de 128 bits (16 octets) et chacun de ces blocs subira un chiffrement pour ensemble former le ciphertext. Dans le cas de l'AES en mode CBC, nous utilisons ce que l'on appelle un IV (Initialization Vector ou vecteur d'initialisation) qui vient réaliser un XOR (ou exclusif) sur le premier bloc du message. Pour les blocs suivants, on viendra réaliser un XOR entre un bloc de message et le ciphertext précédent avant d'utiliser la fonction de chiffrement. [19]

La figure 4 représente cet enchaînement :

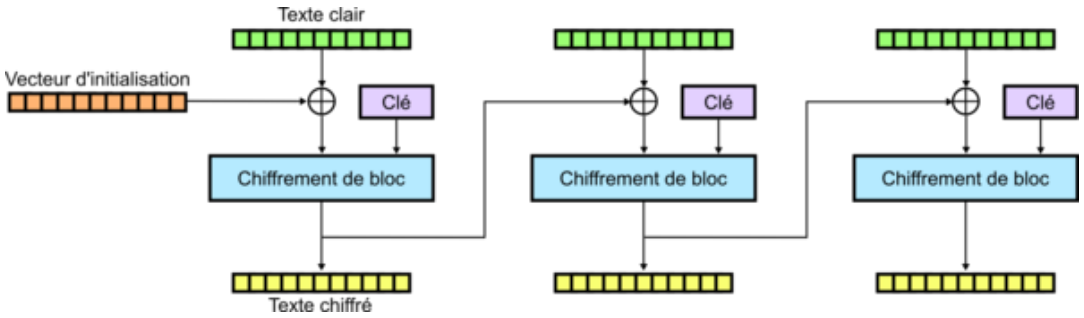


Figure 4: Chiffrement par algorithme AES en mode CBC [19]

L'objectif de cette manière de procéder est d'obtenir deux ciphertext différents pour deux blocs de message identiques afin d'éviter qu'un tiers ne puisse retrouver les données, à partir du ciphertext, sans connaître la clé. L'IV n'a pas besoin d'être secret, tout le monde peut y avoir accès sans pour autant qu'il soit facile de retrouver le message sans connaître la clé secrète.
Bien évidemment, pour le déchiffrement, il suffira d'effectuer l'opération inverse, bloc par bloc et en utilisant la fonction de déchiffrement, pour retrouver le message initial.

3.1.2 Cryptographie à clé asymétrique

Cette fois-ci, nous n'utiliserons pas la même clé pour chiffrer et déchiffrer le message. Nous parlons alors de clé privée et de clé publique. Deux possibilités s'offrent à nous : [24]

- Chiffrer avec la clé publique pour que seul le détenteur de la clé privée puisse déchiffrer le message (notion de confidentialité).
- Chiffrer avec la clé privée pour que les personnes déchiffrant avec la clé publique soient certaines de qui a envoyé/chiffré le message, (notion d'authentification).

La première méthode abordée ci-dessus pourrait par exemple servir pour s'échanger une clé secrète et continuer la communication par chiffrement symétrique pour par exemple profiter de ses meilleures performances.
Nous ne développerons pas plus ce type de cryptographie, car elle sort du cadre de ce stage.

3.2 Autres éléments

Les techniques de cryptographie ne sont aujourd'hui pas utilisées seules, nous devons également y intégrer d'autres méthodes afin de combler les failles potentielles de ces dernières. Cette section sera consacrée à l'explication de deux d'entre elles : Fonction de hachage et padding (remplissage/bourrage en français).

3.2.1 Padding

Comme évoqué dans la section 3.1.1, le chiffrement se fait sur des morceaux de données de taille déterminée (128 bits). Si la taille des données n'est pas exactement un multiple de 128 bits, il faudra ajouter des bits pour faire en sorte qu'elle le devienne. C'est ce que l'on appelle le padding, ou encore le remplissage. Il existe différentes manières d'effectuer un padding, par exemple simplement ajouter des 0 jusqu'à obtenir la taille souhaitée, mais ce n'est pas forcément le choix le plus judicieux. En effet, lors du déchiffrement, il faudra également être capable de distinguer le padding de la donnée pour pouvoir resituer correctement le message initial. [25]

3.2.2 Fonction de hachage

Une fonction de hachage sert, à partir d'une donnée, à produire un hash (ou haché de données) de taille fixe dont la valeur diffère suivant la fonction utilisée et ayant pour but de garantir l'intégrité de la donnée. Elle n'est pas considérée comme une technique de chiffrement car elle n'est pas réversible, c'est-à-dire que nous ne pouvons pas retrouver la donnée à partir du hash. [22]
En revanche, elle peut être utilisée de manière complémentaire à une technique de chiffrement. Par exemple, dans le cas du chiffrement AES CBC, une technique de cryptanalyse appelée "Padding Oracle Attack" [7] consiste à modifier la fin de chaque bloc de ciphertext dans le but de pouvoir extraire le message sans connaître la clé secrète.
Pour éviter cela, une solution consisterait à joindre au ciphertext le hash de ce dernier, et, lors du déchiffrement, recalculer ce hash afin de vérifier si le ciphertext n'a pas été modifié.

3.3 Produits existants dans l'entreprise

Thales propose déjà une solution de chiffrement de donnée à l'aide d'un token USB nommé *Bermudes*. Celui-ci a donc pour but de générer les différentes clés, gérer l'accès à celles-ci par l'intermédiaire d'un mot de passe (code PIN) ainsi que de chiffrer/déchiffrer des données.
Ce token permet donc, par le biais d'une interface utilisateur, de chiffrer/déchiffrer des données sans se préoccuper du fonctionnement de la cryptographie. L'utilisateur peut alors par exemple stocker des fichiers secrets ou encore envoyer un e-mail chiffré à un utilisateur qui possède la même clé.

Partie 4

Tâches réalisées durant le période de stage

Après ces explications théoriques, nous pouvons maintenant nous attarder sur les différentes tâches réalisées durant la période de stage, ainsi que l'explication des besoins de l'entreprise.

4.1 Besoins de l'entreprise

L'entreprise Thales a pour souhait de transposer le projet *Bermudes* au monde des téléphones mobiles. Ceux-ci, ne disposant pas d'un port USB standard, ne peuvent pas directement utiliser le token déjà existant en adaptant simplement l'interface utilisateur en version application mobile. En revanche, la plupart des téléphones Android disposent d'un port microSD.

L'objectif de ce stage est donc de trouver un moyen d'adapter une solution de chiffrement de données à l'univers mobile d'Android, en utilisant une smart-card au format microSD. À terme, ce token devra pouvoir communiquer avec un ordinateur pour bénéficier de la gestion des clés de chiffrement du projet *Bermudes* et également avec un téléphone Android pour réaliser diverses opérations cryptographiques.

Pour pouvoir communiquer avec *Bermudes*, il faudra développer une interface capable de comprendre des opérations définies par le standard PKCS#11 (Public-Key Cryptography Standard #11). PKCS#11 est un standard qui définit la manière d'interagir avec des tokens cryptographiques, par exemple quels sont les différentes méta-données associées à une clé, quelles opérations peuvent être effectuées ou encore comment interagir avec le token. [12]

4.2 Documentation

Une des tâches principales de ce stage a été la recherche et la documentation, notamment sur ce que proposent les différents fabricants de smart-card, quelles sont les différentes fonctionnalités proposées par celles-ci et si elles correspondent aux besoins de l'entreprise.

La société Gemalto [1], par exemple, propose déjà des smart-cards au format microSD ainsi qu'un middleware Android pour interagir avec celle-ci. Malheureusement, aucun des produits existants ne répond parfaitement aux besoins de Thales. Généralement ils ne respectent pas le standard PKCS#11 et ne peuvent donc pas directement venir interagir avec la gestion des clés déjà présente dans l'entreprise.
La société Gemalto propose en revanche des smart-cards "vides", donc sans firmware, sans applet Java, permettant ainsi de développer nous même les fonctionnalités désirées.

J'ai donc passé le début de mon stage à me documenter sur comment réaliser une applet Java, comment communiquer avec la carte ou encore quelles sont les limitations dues à ce hardware restreint.

4.3 Smart-card

Une smart-card est une carte à puce portant au moins un circuit-intégré capable de contenir de l'information. Elle est constituée d'un microprocesseur pour pouvoir traiter cette information et éventuellement d'un composant de sécurité comme un cryptoprocesseur capable de générer et stocker des clés privées. Elle possède également une mémoire vive RAM (Random Access Memory), une mémoire non-volatile EEPROM (Electrically-Erasable Programmable Read-Only Memory) et dans certains cas une mémoire FLASH de plus grosse capacité. [26]

4.3.1 Communication

Pour communiquer avec une smart-card, il existe un protocole d'échange de message nommé APDU (Application Protocol Data Unit) qui définit la structure des messages entre une carte à puce et un lecteur de carte à puce. Ce protocole est normalisé par l'ISO 7816. [8]
Un message APDU est constitué d'un HEADER et d'un BODY. Le HEADER reprend 4 bytes nommés CLA, INS, P1 ainsi que P2, le BODY reprend 3 parties nommées LC, DATA et LE. Dans l'APDU standard, LC et LE ont une longueur de 1 byte et DATA une longueur maximale de 255 bytes. Cependant, les nouvelles smart-cards peuvent également supporter l'Extended APDU qui permet simplement d'utiliser une taille de DATA allant jusqu'à 65 535 bytes. Si la donnée

[1] Gemalto est un fournisseur de produits de sécurité https://www.gemalto.com/france

DATA excède 255, LC doit passer de 1 byte à 3 bytes et LE de 1 byte à 2 bytes. Le tableau 1 reprend la signification de chacune des parties d'un message APDU.

Nom	**Longueur (bytes)**	**Description**
CLA	1	Classe d'instruction indiquant le type de commande, par example 'interindustry' ou 'proprietary'
INS	1	Code d'instruction
P1-P2	2	Paramètres d'instructions pour la commande
LC	0, 1 ou 3	Définit le nombre de bytes de DATA envoyé par la commande
DATA	0...65535	DATA envoyée de longueur LC
LE	0, 1 ou 2	Définit le nombre de bytes attendus dans la réponse

Tableau 1: Structure d'une commande APDU [8]

Lorsque la commande APDU a été traitée par la smart-card, elle va pouvoir répondre à la demande via une réponse APDU. Celle-ci est divisée en 3 parties : DATA de longueur maximale LE précisée dans la commande, SW1 ainsi que SW2 ayant pour signification :

Nom	**Longueur (bytes)**	**Description**
DATA	0...65535	DATA reçue de longueur maximale LE
SW1	1	Byte de statut numéro 1
SW2	1	Byte de statut numéro 2

Tableau 2: Structure d'une réponse APDU [8]

Les deux bytes de statut SW valent respectivement 90 et 00 en hexadécimal lorsque la smart-card a su traiter la commande sans rencontrer de problème. Dans le cas échéant, ils peuvent valoir deux types de messages : Un statut d'erreur propre au protocole APDU ou un statut défini dans la Java applet de la carte. Les statut d'erreur d'APDU peuvent par exemple être : 67 00 (Wrong length) lorsque la taille de la donnée envoyée n'est pas la valeur précisée dans le champ LC, 69 F0 (Permission Denied) quand on essaye d'envoyer une commande APDU nécessitant certains privilèges ou encore 6E 00 (Class not supported) lorsque le champ CLA est invalide. [6]

4.3.2 Environnement de développement

Afin de simplifier la programmation des cartes à puce, en 1997 les sociétés Schlumberger et Gemplus fusionnent pour fonder JavaCard, un système d'exploitation pour smart-card permettant de développer des applications

génériques capables de tourner sur tout type de carte. En effet, avant JavaCard, les applications devaient être codées de manière spécifique pour chaque fournisseur de carte. [23]
JavaCard devient très vite la plate-forme standard de développement des applications pour smart-card. Aujourd'hui, JavaCard comprend un toolkit nommé JCDK (Java Card Development Kit) fournissant notamment un simulateur de carte nommé CREF, un outil de transformation de code Java en applet JavaCard ou encore un outil d'envoi de commandes APDU entre une console et un simulateur. [10]
Le toolkit JCDK fournit également un plug-in lui permettant de facilement s'intégrer dans un IDE, par exemple Eclipse d'Oracle que j'ai utilisé durant mon stage.

Pour l'intégration de l'applet sur une carte physique, nous avons également besoin de GlobalPlatformPro, un outil open source de gestion d'applet et de clés sur smart-card. Notons également qu'un smart-card peut contenir plusieurs applets différentes, mais que une seule ne peut être active (sélectionnée) à la fois.

4.3.3 Applet

Une applet Java est un programme fourni sous la forme bytecode Java. Une applet JavaCard est une applet respectant la norme ISO 7826 précédemment introduite simplifiant le développement d'application JavaCard par encapsulation des commandes bas niveaux d'envoi et de réception de messages APDU dans un framework Java.
Une Applet JavaCard doit alors étendre *javacard.framework.Applet* qui définit les méthodes que doit supporter une applet pour pouvoir interagir avec le JCRE (Java Card Runtime Environnement) d'une smart-card. Dans ces méthodes on retrouve quelques exemples cités dans le tableau 3. [23]

Nom de la méthode	**Description**
install(byte[] bArray, short bOffset, byte bLength)	Appelé par le JCRE pour créer une instance de l'applet.
select()	Appelé par le JCRE pour informer l'applet que celle-ci a été sélectionnée.
process(APDU apdu)	Appelé par le JCRE lors de la réception d'une commande APDU.
deselect()	Appelé par le JCRE pour informer l'applet sélectionnée qu'une autre applet va être sélectionnée et que celle-ci ne le sera plus

Tableau 3: Exemple de méthode définie par *javacard.framework.Applet* [29]

Il existe quelques contraintes complémentaires liées lors du développement d'applet JavaCard venant du fait que l'applet tournera sur un hardware restreint. Par exemple, les types de données simples utilisés dans le programme sont limités

à boolean, short, byte et byte array. Il faut également prendre en compte le fait que le ramasse-miette (appelé Garbage Collector en anglais) ne fonctionne pas de la même façon que dans un environnement Java classique. En effet, dans JavaCard, les objets sont stockés dans la mémoire non-volatile EEPROM ayant un temps d'accès non négligeable ce qui entrainerait une perte de performance lors de chaque invoquation du ramasse-miette. Il faudra donc l'invoquer soi-même en faisant un compromis entre perte de performance et manque de mémoire.

Il est également possible de travailler avec la mémoire RAM de la carte à puce en précisant cette fois que l'objet doit y être stocké. Cependant, la taille de cette mémoire est encore plus restreinte que celle de l'EEPROM, nous sommes alors encore plus limités dans la taille des données que l'on traite. Une bonne pratique de la programmation JavaCard est d'instancier les différents objets dans le constucteur de l'applet (voir methode *install* du tableau 3) avec une taille fixe. Par exemple, un buffer représenté par un array de bytes, devra dans le constructeur posséder une taille maimale, et il faudra travailler en plusieurs passes si besoin d'une plus grande taille. [11]

4.3.4 Hardware

Lors de mon stage j'ai utilisé un smart-card IDCore 8030 [3] de la société Gemalto, vide donc sans applet comme précisé dans la section 4.2. Cette carte au format microSD possède une mémoire RAM de 8Ko, une EEPROM de 80Ko, ainsi qu'une mémoire flash de 8Go. Malheureusement, la carte ne supporte pas le mode Extended d'APDU car elle possède un buffer hardware de 255 bytes pour la récéption de commande APDU. Cette contrainte nous limite à un champ de donnée de taille maximale de 240 bytes pour laisser de la place à la partie HEADER de la commande.

4.4 Développement

Maintenant que le contexte des smart-cards est introduit, il est possible de s'intéresser au travail effectué durant le stage, à savoir, le développement d'une première version d'une applet JavaCard pour réaliser du chiffrement de donnée ainsi que d'un middleware capable de communiquer avec la carte et s'interfacer avec le standard PKCS#11. Il s'agit donc d'une structure en couche telle que décrite ci-dessous:

Couche 0 Hardware, effectue les différentes opérations

Couche 1 Firmware, reçoit des instructions et données via APDU, les traite et renvoie via ce même protocole

Couche 2 Middleware, recoit des instructions et données via PKCS#11 et les transmet via APDU au firmware

Couche 3 Application, recoit les entrées de l'utilisateur et les transmet via PKCS#11 au middleware

Le hardware ayant déjà été introduit dans la section 4.3.4, seule l'analyse la structure des autres couches sera décrite dans les sections suivantes. Afin de ne pas surcharger les explications ainsi que les diagrammes, seules les méthodes étant nécessaires à la compréhension générale seront mentionées.

4.4.1 Structure de firmware

Le firmware, ou applet JavaCard, a pour responsabilité le stockage des clés de manière sécurisée sur le hardware avec toutes leurs méta-données. C'est également lui qui s'occupe de la recherche de clé sur base de méta-données et finalement la réalisation des opérations de chiffrement et de déchiffrement.

La structure selon laquelle a été réalisé le firmware exposé à la figure 5 est constituée d'une classe principale servant de point d'entrée pour les commandes APDU. Ce point d'entrée est obtenu grâce à l'extension de la classe *javacard.framework.Applet* comme énoncé dans la section 4.3.3. Cette classe nommée *BermudesApplet* permet donc de traiter les commandes APDU et de réaliser une instruction spécifique par le biais d'un switch-case sur le champ INS de la commande reçue. C'est notamment cette classe qui va réaliser les opérations de chiffrement et de déchiffrement.
Bien que la smart-card utilisée durant ce stage ne supporte pas l'Extended APDU, cette classe implémente également l'interface *ExtendedLength* [13] qui permet son utilisation afin de prévoir l'éventualité d'un changement de support hardware.

S'ajoutent à cette classe deux autres classes nommées *KeyObject* et *TemplateHandler* permettant respectivement le maintient des clés ainsi que leurs méta-données et chercher si une clé correspond à un template fourni. Un template est simplement une liste de valeur d'attribut à rechercher parmi les méta-données de l'ensemble des clés présentes sur la carte.

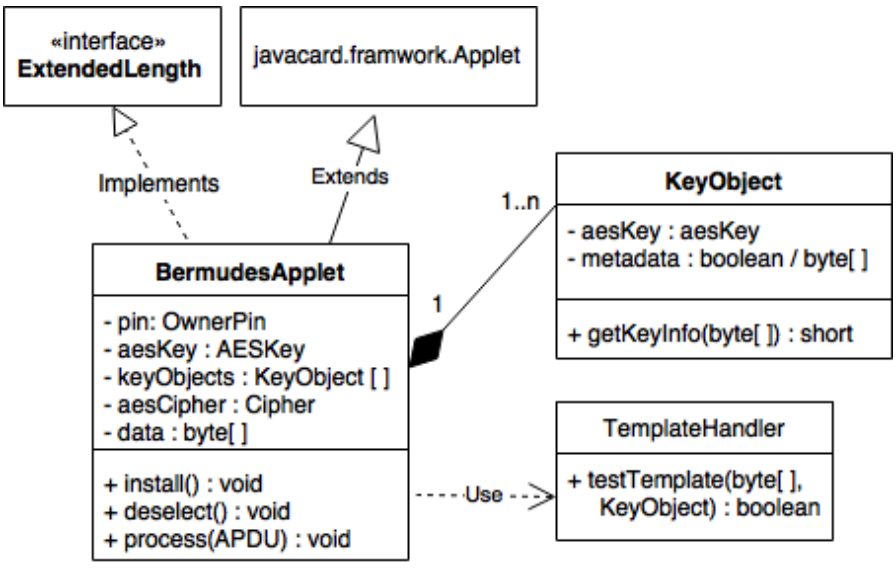


Figure 5: Structure de l'applet Javacard [2]

4.4.2 Structure du middleware

Le middleware a quant à lui pour responsabilité de transformer une requête provenant d'une application en une commande APDU interprétable par le firmware. Il recevra alors une réponse APDU qu'il mettra en forme pour l'application. Rajoutons également le fait que ces requêtes provenant de l'application respectent le standard PKCS#11.

La figure 6 représente la structure de ce middleware. La classe principale, *RequestHandler*, contient toutes les méthodes pouvant être appelées par une application, comme *encryptFile* qui prend en input un array de bytes provenant d'un fichier, et ressort un nouvel array de bytes représentant le ciphertext calculé par le firmware.
Notons que cette classe est responsable de la connexion à la smart-card s'effectuant après avoir reçu un objet CardTerminal [9] en paramètres de son constructeur. L'instanciation de cette classe est alors restreinte à un seul objet via le design pattern *Singelton* afin d'éviter d'ouvrir plusieurs connexions en même temps sur la carte.

La classe nommée *DataHandler* fournit simplement des fonctions servant à traiter des données. Prenons par exemple *padData* qui s'occupera de faire le padding des données, nécessaire lors d'un chiffrement par bloc tel qu'AEC CBC, comme proposé dans la section 3.2.1. Cette fonction s'occupe d'effectuer le padding de données selon la méthode définie par PKCS#5, c'est-à-dire en ajoutant à la fin du message un certain nombre n de bytes b jusqu'à ce que la taille de la donnée soit un multiple de 16. La valeur de ces bytes b vaut le nombre de byte ajouté n. Si la donnée est déjà un multiple de 16, on ajoutera alors simplement 16 bytes de valeur 16.
Cette classe permet également de supprimer le padding, formatter des listes d'objet en array de bytes ou encore de générer un IV pour le chiffrement AES.

Nous retrouvons encore deux classes, une nommée *MetaAttribute* qui définit simplement la structure de chaque attribut d'une clé secrète comme défini par le standard PKCS#11 et l'autre, *ApduExcpetion*, qui permet de générer des exceptions lorsque le statut SW de la réponse APDU est différent de 90 00. Cette dernière fera alors un lien entre le code d'erreur reçu et le message d'erreur à afficher à l'utilisateur. Nous avions introduit dans la section 4.3.1 les deux types de messages d'erreur possibles à savoir, propre à APDU ou définit dans l'applet. Nous pouvons citer quelques exemples de code d'erreur défini dans le firmware tel que : 00 06 lorsque l'instruction INS n'existe pas, 00 32 lorsque le code PIN spécifié par l'utilisateur n'est pas correct ou encore 00 09 lorsque l'utilisateur essaie d'accéder à du contenu protégé.

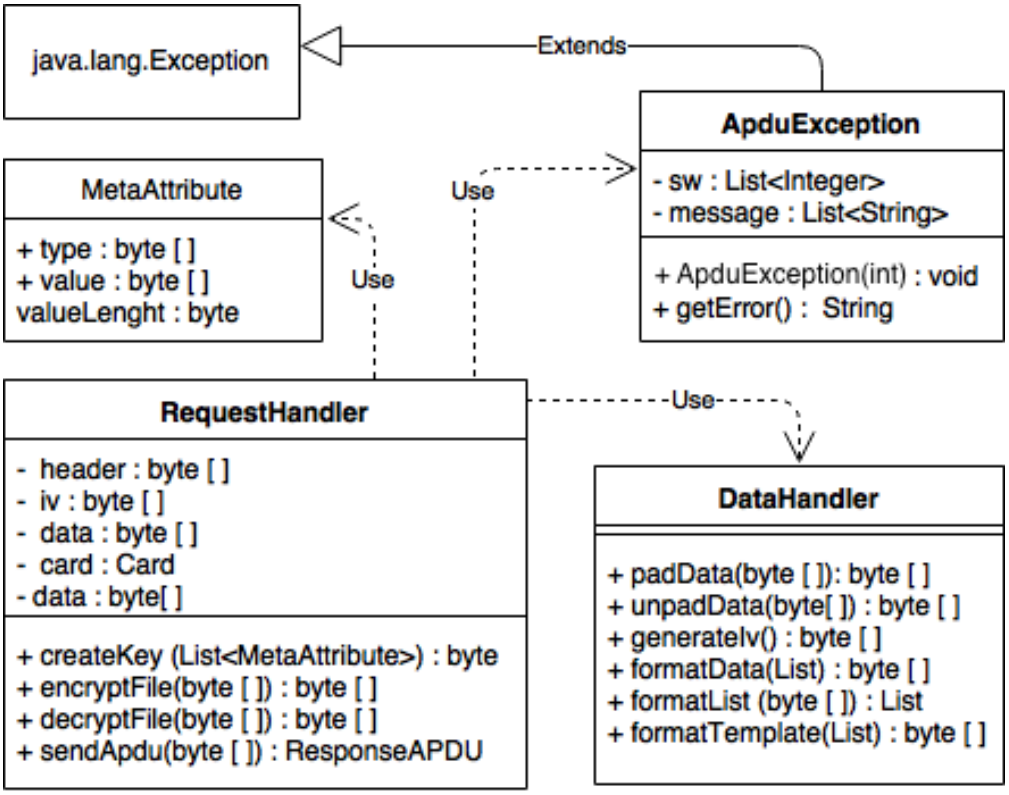


Figure 6: Structure du middleware [2]

4.4.3 Couche applicative

Finalement, une petite application a été développée au-dessus de ce midlleware. Cette application, fort simpliste, n'as pas de structure particulière. Dans un premier temps celle-ci n'est qu'une application console permettant à l'utilisateur de créer des nouvelles clés en entrant directement la valeur de celles-ci, ou encore de sélectionner un fichier à chiffrer/déchiffrer.

Cette petite application ne sera utilisée que pour pouvoir réaliser une démonstration ainsi que quelques tests de vitesse de chiffrement de gros fichier. L'application vient alors juste envoyer des données au midlleware sous une forme ressemblant à celle définie dans PKCS#11 et affiche à l'utilisateur la réponse reçue.

4.5 Résultats et interprétation

L'application console qui interagit avec le middleware proposé à la figure 7 fonctionne en plusieurs étapes. Si l'utilisateur veut simplement chiffrer un fichier, voici le déroulement standard :

1. Recherche des smart-cards connectées à l'ordinateur, et demande à l'utilisateur de choisir la carte voulue. Connexion automatique à la carte si une seule est présente.

2. L'utilisateur doit ensuite rentrer son code PIN, qu'il peut également changer par la suite.

3. Si aucune clé n'est présente sur la carte, il faut en ajouter une. Nous récupérerons alors l'object handler (référence d'une clé pour le middleware) de cette nouvelle clé. Sinon, passer cette étape.

4. Si le client ne connait pas l'object handler de la clé souhaitée, il peut effectuer un *Search template* pour trouver l'object handler qui répond à certains critères. Dans le cas contraire passer cette étape.

5. Il faut maintenant utiliser *Set key for crypto* en précisant l'object handler pour que le firmware sache avec quelle clé il devra effectuer les prochaines opérations cryptographiques.

6. Nous pouvons maintenant utiliser l'option *Encypt file* qui demande à l'utilisateur de choisir le fichier à chiffrer.

7. L'application sauve le ciphertext au même endroit que le fichier, avec le même nom, en y ajoutant l'extension *.crypt*.

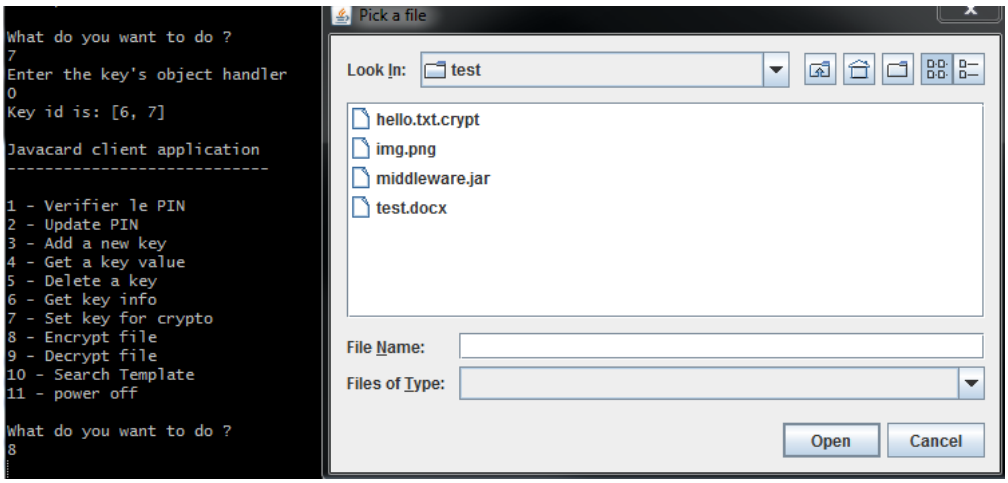


Figure 7: Capture d'écran de l'application en cours de fonctionnement [2]

Notons que pour la partie déchiffrement, le ciphertext contient l'identifiant de clé avec laquelle il a été chiffré, il ne faudra donc pas que l'utilisateur précise cette dernière. En revanche il faut que celle-ci soit bien présente sur la smart-card sinon l'utilisateur ne sera pas en mesure de déchiffrer le fichier.

D'autres opérations sont également possibles, citons par exemple *Delete a key* qui supprime une clé de la carte, *get a key value* qui renvoie la clé en l'utilisateur si celle-ci n'est pas sensitive (méta-donnée associée à la clé) ou encore *Get key info* qui renvoie à l'utilisateur un tableau contenant toutes les méta-données d'une clé.

Coté middleware, si le fichier à encrypter est trop volumineux, il sera découpé en plusieurs parties de 224 bytes pour respecter la limitation du hardware en termes

de taille de message APDU tel qu'exposé dans la section 4.3.4. La limite hardware étant de 255 bytes, si nous gardons quelques bytes pour HEADER APDU ainsi que 16 bytes pour l'IV d'AES CBC, il n'en reste que 224 pour la donnée à chiffrer. Cette découpe va alors engendrer une limitation des performances de l'application à 2,5 kbytes/seconde de moyenne pour le chiffrement. Plusieurs tests ont été réalisés en diminuant encore la taille des blocs de fichier afin d'en évaluer l'impact sur les performances. Le résultat de ces tests exposé à la figure 8 nous indique clairement qu'augmenter la taille des blocs envoyés à la smart-card influe sur le débit de bytes par seconde que l'on peut obtenir.

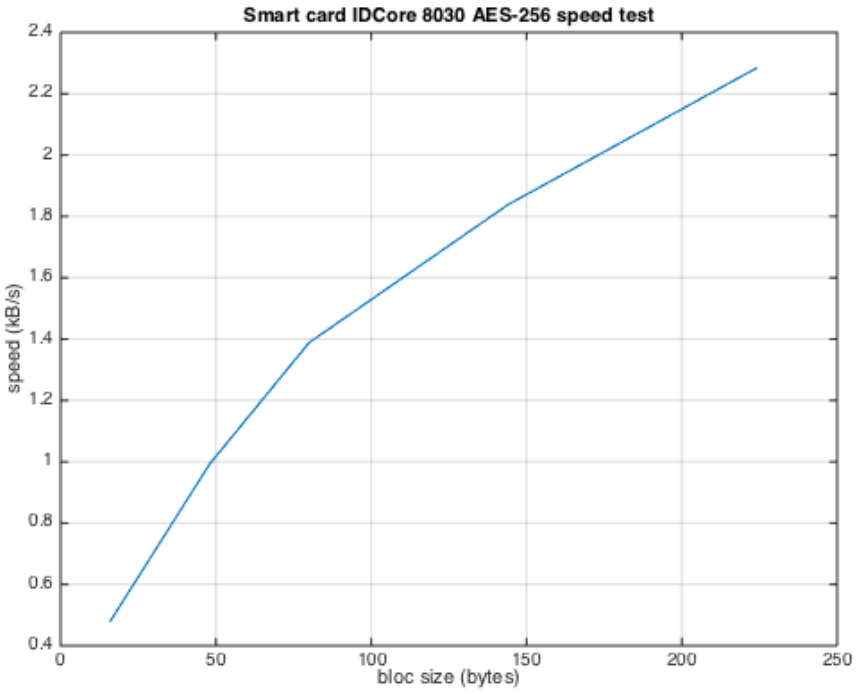


Figure 8: Test de performances en fonction de la taille des blocs [2]

Il pourrait alors être intéressant d'utiliser un support hardware capable de recevoir des messages APDU en mode Extended afin de pouvoir atteindre le débit maximal possible. Nous pouvons supposer qu'augmenter la taille des messages jusqu'à une taille infinie aura également un impact négatif sur les performances, mais il pourrait être intéressant de trouver un point de compromis optimal entre taille des blocs et performances. Notons également que le firmware a été développé de manière à pouvoir recevoir des messages Extended APDU, il n'y aura donc rien à modifier de ce côté la.

Partie 5

Conclusion

Dans l'ensemble, le stage a été pour moi une belle expérience. J'ai été confronté à des sujets complexes et inconnus pour lesquels j'ai dû correctement me documenter afin de comprendre exactement ce qui était attendu. J'ai eu l'occasion de participer à diverses réunions ainsi qu'à mon propre projet, permettant d'améliorer ma vision de la vie d'ingénieur en entreprise.
J'ai également eu l'opportunité d'organiser une réunion de fin de stage, afin de tenir informé le personnel de Thales de l'état d'avancement du projet qui m'avait été confié. Organiser des réunions ou brainstormings est, je pense, une des compétences qu'un ingénieur doit acquérir pour compléter sa formation.

Ce stage a également été pour moi une bonne manière d'appréhender mon travail de fin d'étude portant sur le même sujet, mais ayant pour objectif d'aller plus loin dans le développement de la chaîne complète de l'application. Je pars alors avec une base solide en termes compréhension du fonctionnement de l'environnement Javacard et de ses spécificités.

En se référant à la note d'activités située en annexe de ce document, nous pouvons conclure que chacun des objectifs, tant au niveau insertion dans l'entreprise qu'au niveau technique, a pu être complété au cours de la période de stage.

Bibliographie

[1] ANS. Autorité nationale de sécurité (ans). https://diplomatie.belgium.be/fr/sur_lorganisation/organigramme_et_structure/ans, 2017.

[2] Antoine, V. M. Document réalisé en interne, 2017.

[3] Bonnet, D. Idprime md 8840 and idcore 8030microsd cards. http://studylib.net/doc/5813123/idprime-md-8840-and-idcore-8030, 2015.

[4] Bourgeois, M. Initiation à pgp : Gnupg. http://mbourgeois.developpez.com/articles/securite/pgp/, 2006.

[5] CommentCaMarche. Introduction au chiffrement avec DES. http://www.commentcamarche.net/contents/204-introduction-au-chiffrement-avec-des, 2017.

[6] EFTlab. Complete list of apdu responses. https://eftlab.co.uk/index.php/site-map/knowledge-base/118-apdu-response-list, 2017.

[7] Heaton, R. The padding oracle attack - why crypto is terrifying. https://robertheaton.com/2013/07/29/padding-oracle-attack/, 2013.

[8] Jacquinot Consulting. Iso 7816-4: Interindustry commands for interchange. http://www.cardwerk.com/smartcards/smartcard_standard_ISO7816-4_5_basic_organizations.aspx, 2017.

[9] Oracle. Class cardterminal. https://docs.oracle.com/javase/7/docs/jre/api/security/smartcardio/spec/javax/smartcardio/CardTerminal.html, 2005.

[10] Oracle. Java card platform, classique edition. https://docs.oracle.com/javacard/3.0.5/index.html, 2017.

[11] Ortiz, C. E. An introduction to java card technology - part 1. http://www.oracle.com/technetwork/java/javacard/javacard1-139251.html, 2003.

[12] RSA Laboratories. Cryptographic token interface standard. https://www.cryptsoft.com/pkcs11doc/STANDARD/pkcs-11v2-11r1.pdf, 2001.

[13] Sun Microsystems. Interface extendedlength. https://www.win.tue.nl/pinpasjc/docs/apis/jc222/javacardx/apdu/ExtendedLength.html, 2005.

[14] Thales Belgium. Documentation interne thales belgium, 2017.

[15] Thales Group. Chiffres clés. https://www.thalesgroup.com/fr/investisseurs/chiffres-cles, 2015.

[16] Thales Group. Cryptographic equipments. https://www.thalesgroup.com/en/cryptographic-equipments, 2017.

[17] Thales Group. Thales group. https://www.thalesgroup.com, 2017.

[18] Vandevoodre, C., and Vander Meiren, A. Document réalisé en interne, 2017.

[19] Wikipédia. Block cipher mode of operation. https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation, 2016.

[20] Wikipédia. Advanced encryption standard. https://fr.wikipedia.org/wiki/Advanced_Encryption_Standard, 2017.

[21] Wikipédia. Cryptographie symétrique. https://en.wikipedia.org/wiki/Symmetric-key_algorithm, 2017.

[22] Wikipédia. Fonction de hachage cryptographique. https://fr.wikipedia.org/wiki/Fonction_de_hachage_cryptographique, 2017.

[23] Wikipédia. Java card. https://fr.wikipedia.org/wiki/Java_Card, 2017.

[24] Wikipédia. Public-key cryptography. https://en.wikipedia.org/wiki/Public-key_cryptography, 2017.

[25] Wikipédia. Remplissage (cryptographie). https://fr.wikipedia.org/wiki/Remplissage_(cryptographie), 2017.

[26] Wikipédia. Smart card. https://en.wikipedia.org/wiki/Smart_card, 2017.

[27] Wikipédia. Thales. https://fr.wikipedia.org/wiki/Thales, 2017.

[28] Wikipédia. Thomson-houston electric company. https://fr.wikipedia.org/wiki/Thomson-Houston_Electric_Company, 2017.

[29] Zhiqun, C. How to write a java card applet: A developer's guide. https://www.javaworld.com/article/2076450/client-side-java/how-to-write-a-java-card-applet--a-developer-s-guide.html, 1999.

Annexe

Voici la note d'activités complétée en début de stage, elle contient notamment les différents objectifs de celui-ci.

Maître de stage : *TESSIAN Samuel* samuel.tessian@be.thalesgroup.com *0471 47 07 55*
Date : 20/09/2017
Superviseur : COMBEFIS Sébastien cbf@ecam.be
Etudiant : *VANDER MEIREN Antoine* antoine.vdm@icloud.com *0490 43 07 99*

Entreprise : THALES Belgium s.a., Rue des Frères Taymans 28, B-1480 TUBIZE
Période de stage : du 15/09/2017 au 27/10/2017

NOTE D'ACTIVITES

Objectifs généraux	Objectifs spécifiques
a) Insertion dans l'entreprise	- Intégration dans une équipe d'ingénieur - Participation aux réunions planifiées de suivi
b) Mener à bien un projet	- Recherche des différentes possibilités de développement d'un firmware pour smart-card - Développement de l'applet JavaCard servant à réaliser du chiffrement de données - Implémentation du firmware sur un hardware physique - Réalisation d'un middleware pour communiquer avec la carte

Signature de l'étudiant	Signature du Maître de stage
Vander Meiren Antoine	TESSIAN Samuel